

== Messages from file: [BBN=TENEXE]<MCKENZIE>MESSAGE.TXT;1
-- Wednesday, July 20, 1977 09:37:10-EDT ==

144 == *****

Date: 19 Jul 1977 1839-EDT

From: WALDEN

Subject: fyi

To: Bressler, Heart, McKenzie, McQuillan, Walden

Mail from USC-ISIE rcvd at 19-JUL-77 1828-EDT

Date: 19 Jul 1977 1523-PDT

From: POSTEL at USC-ISIE

Subject: material for arpanet completion report

To: Walden at BBNE

cc: postel at ISIB

Dave:

the following information may be of assistance in your work on the arpanet report, if i can be of any assistance in tracking down dates of meetings or publications, or any other help let me know,

--Jon,

Chronology

April 1969

Network Working Group formed, First RFC published,

S. Crocker "Host Software," RFC 1, NIC 4687, 7-Apr-69,

May 1969

First host level Protocol Specified,

G. Deloche "Host Software," RFC 9, NIC 4695, 1-May-69,

February 1970

Second host level protocol proposed,

S. Crocker "New HOST=HOST Protocol," RFC 33, NIC 4735,

12-Feb-70,

June 1970

Version 2 of second host level protocol proposed,

S. Crocker, "An Official Protocol Proffering," RFC 54, NIC

4756, 18-Jun-70,

August 1970

Host level protocol published as official specification,

S. Crocker, "Official Host to Host Protocol," Document 1,

NIC 7149, 3-Aug-70,

The Message Switching host level protocol is proposed,

D. Walden "A System for Interprocess Communication in a

Resource Sharing Computer Network," RFC 62, NIC 4962,

3-Aug-70, Also published in Communications of the ACM,

15(4):221-230, April 1972,

Network Graphics Meeting

Where, Who ???

Initial Connection Protocol proposed,

S. Crocker "3rd Level Ideas and other Noise," RFC 66, NIC

5409, 26-Aug-70,

November 1970

Network Working Group Meeting at FJCC in Houston,

J. Postel "Network Meeting Report," RFC 77, NIC 5604,

20-Nov-70,

E. Meyer "Network Meeting Notes," RFC 82, NIC 5619,

9-Dec-70,

Initial network use at UCSB and RAND,

E. Harslem "NCP Status Report: UCSB/RAND," RFC 78, NIC 5199,

undated,
 December 1971
 Data Reconfiguration Service first suggested,
 E. Harslem "Protocols and Data Formats," RFC 80, NIC 5608,
 1=Dec=70,
 January 1971
 Graphics Protocol first proposed,
 S. Crocker "Proposal for a Network Standard Format for a
 Data Stream to Control Graphics Display," RFC 86, NIC 5631,
 5=Jan=71,
 Remote Job Entry protocol first proposed,
 R. Braden "NETRJS - A Third Level Protocol for Remote Job
 Entry," RFC 88, NIC 5668, 13=Jan=71,
 Initial use of the network at MIT and Harvard,
 R. Metcalfe "Some Historic Moments in Networking," RFC 89,
 NIC 5697, 19=Jan=71,
 February 1971
 Telnet protocol first proposed,
 J. Melvin "A First Cut at a Proposed Telnet Protocol," RFC
 97, NIC 5740, 15=Feb=71,
 E. Meyer "Logger Protocol Proposed," RFC 98, NIC 5744,
 11=Feb=71,
 Network Working Group held at University of Illinois,
 R. Watson "Notes on the Network Working Group Meeting," RFC
 101, NIC 5762, 23=Feb=71,
 R. Watson, "Addendum to NWG Meeting Notes," RFC 108, NIC
 5807, 25=Mar=71,
 Host level protocol modified,
 S. Crocker "Output of the Host/Host Protocol Glitch Cleaning
 Committee," RFC 102, NIC 5763, 22=Feb=71,
 March 1971
 Host level protocol modified,
 R. Bressler "Output of the Host=Host Protocol Glitch
 Cleaning Committee," RFC 107, NIC 5806, 23=Mar=71,
 April 1971
 File Transfer Protocol first proposed,
 A. Bhushan "A File Transfer Protocol," RFC 114, NIC 5823,
 16=Apr=71,
 May 1971
 Network Working Group meeting held at SJCC in Atlantic City,
 J. Heafner, "Minutes of the Network Working Group Meeting,"
 RFC 164, NIC 6778, 25=May=71,
 Telnet protocol specified,
 T. O'Sullivan "TELNET Protocol," RFC 158, NIC 6768,
 19=May=71,
 Data Reconfiguration Service specified,
 B. Anderson "Data Reconfiguration Service == An
 Implementation Specification," RFC 166, NIC 6780, 25=May=71,
 June 1971
 File Transfer Protocol design initiated,
 Bhushan, A. "The Data Transfer Protocol," RFC 171, NIC 6793,
 23=JUN=71,
 Bhushan, A. "The File Transfer Protocol," RFC 172, NIC 6794,
 23=JUN=71,
 July 1971
 Mail Protocol first discussed,
 Watson, R. "A Mail Box Protocol," RFC 196, NIC 7141,
 20=JUL=71,
 Graphics meeting held by MIT,
 August 1971
 Terminal IMP Protocol Implementation
 McKenzie, A. "NCP, ICP, and Telnet: The Terminal IMP

Implementation," RFC 215, NIC 7545, 30-AUG-71.
 October 1971
 Network Working Group Meeting and Programmers Workshop
 Vezza, A. "Network Working Group Meeting Schedule," RFC 234,
 NIC 7651, 5-OCT-71.
 Postel, J. "Report of the Protocol Workshop," RFC 295, NIC
 8335, 2-JAN-72.
 November 1971
 Network Graphics Meeting
 Raditsky, M. "Graphics Meeting Report," RFC 282, NIC 8164,
 8-DEC-71.
 January 1972
 Protocol proposed by Graphics Committee.
 Michener, J. "Graphics Protocol - Level 0 only," RFC 292,
 NIC 8302, 12-JAN-72.
 April 1972
 File Transfer protocol Meeting
 Bhushan, A. "Data and File Transfer Workshop Notes," RFC
 327, NIC 9634, 17-MAR-72.
 Graphics protocol meeting
 Teinet Protocol specification published
 Postel, J. "Teinet Protocol," RFC 318, NIC 9348, 3-APR-72.
 June 1972
 Remote Job Entry protocol published
 Holland, C. "Proposed Remote Job Entry Protocol," RFC 360,
 NIC 10602, 24-JUN-72.
 August 1972
 Mail commands included in evolving File Transfer protocol
 specification
 Bhushan, A. "Comments on the File Transfer Protocol (RFC
 354)," RFC 385, NIC 11357, 18-AUG-72.
 October 1972
 Demonstration of ARPANET at International Computer
 Communications Conference (ICCC) in Washington D.C.
 January 1973
 The principle of negotiated options for teinet first
 announced.
 Cosell, B. "TELNET Issues," RFC 435, NIC 13675, 5-JAN-75.
 February 1973
 A common user command language proposed.
 Raditsky, M. "Alternative Proposal for a Unified User Level
 Protocol," RFC 451, NIC 14135, 22-FEB-73.
 Mail protocol meeting
 Kudlick, M. "Network Mail Meeting Summary," RFC 469, NIC
 14798, 8-MAR-73.
 Bhushan, A. "FTP and Network Mail System," RFC 475, NIC
 14919, 6-MAR-73.
 March 1973
 Teinet Protocol Meeting
 Mckenzie, A. "Teinet Protocol Meeting Announcement," RFC
 461, NIC 14416, 14-FEB-73.
 File Transfer protocol Meeting
 Mckenzie, A. "File Transfer Protocol Meeting Announcement
 and a Proposed Document," RFC 454, NIC 14333, 16-FEB-74.
 May 1973
 Teinet protocol specified, a thorough revision of the control
 signal mechanisms and the adoption of the negotiated option
 concept.
 Mckenzie, A. "Teinet Protocol Specification," RFC 495, NIC
 15371, 1-MAY-73
 Resource Sharing Executive Workshop meeting

Users Interest group meeting

Crocker, D. "Arpanet Users Interest Working Group Meeting,"
RFC 585, NIC 20050, 6=NOV=73,

June 1973

A file access protocol is proposed,

Day, J. "File Access Protocol," RFC 520, NIC 16819,
25=JUN=73,

July 1973

Network Graphics Meeting

Bunch, S. "Minutes of Network Graphics Group Meeting," RFC
549, NIC 17795, 21=AUG=73,

October 1973

A common text editing subsystem proposed,

Padlipsky, M. "Neted: A Common Editor for the ARPA Network,"
RFC 569, NIC 18972, 15=OCT=73,

July 1974

A cross-network debugging program is proposed,

Mader, E. "Network Debugging Protocol," RFC 643, NIC 30873,
JUL=74,

November 1974

A Host to Front End protocol proposed,

Padlipsky, M. "A Proposed Protocol for Connecting Host
Computers to ARPA-Like Networks via Directly-Connected Front
End Processors," RFC 647, NIC 31117, 12=NOV=74,

December 1974

A procedure oriented protocol is proposed,

Postel, J. "Procedure Call Protocol Documents," RFC 674, NIC
31484, 12=DEC=74,

June 1975

Major extensions to the IMP/Host interface proposed

Walden, D.C. "IMP/Host and Host/IMP Protocol Change," RFC
687, NIC 32654, 2=JUN=75,

Proposal for a Network Interchange Language

Introduction

In this paper an attempt is made to specify a high level programming language for computer networks, and more specifically the ARPA network. The main concept introduced is the one of an abstract Network Machine, which is consistent with the idea of a HOST asking a service from the computer network considered as an overall computing facility. The dialogue is always between a HOST and the Network Machine which language is always the same, though its configuration may vary according to the real remote HOST.

From a programming language point of view, this concept is similar to the UNCOL proposed in 1958 [STR058] but never implemented, however; the application to a computer network implies a realtime interaction between programs. Also, the possibility for the user to use NIL either in a standard mode or in an extended mode where he defines himself his own entities should give to NIL a maximum of flexibility.

1. Basic concepts introduced in NIL

1.1 Aim of NIL

The two main objectives of NIL are:

- a) to describe the environment in which a program is executed (its complement); this involves the description of:
 - data formats and data structures
 - exchanges with input and output devices and characteristics expected from them
 - interface with operating system
- b) to express the front end part of an interactive system:

The data flow through an interactive system generally decreases as the data reaches the kernel of the system: it is assumed that in many interactive systems a separable module exists or can be defined which involves a great amount of data exchanged with the user, and much less exchanged with the rest of the system. This module is called Front-End. It is important that the response time of the system is affected as little as possible by additional transmission delays. Also, it is desirable to keep the data rates as low as possible on the network.

It is assumed that the transfer of a Front End does not imply to solve the whole problem of program transferability.

1.2 NIL subcategorization

As pointed out by S. Volansky [] it's convenient to divide languages in several sublanguages corresponding to their main functions. NIL is thus subdivided in:

- a control sublanguage
- an operation sublanguage
- a data declaration sublanguage
- an environment sublanguage

1.2.1 Control sublanguage

The control sublanguage states WHEN a computation is done: It describes the flow of control or ordering of the computations. With some information contained from the other sections of the language, it also states WHERE the computations are to be executed.

As a computer network introduces loose connections between several systems, the control language of the Network Machine should be able, in an elaborate version, of assigning computations to available processors, taking into account the time delays and resource allocation problems involved. It is not our purpose to consider this level at the moment.

1.2.2 Operation sublanguage

The operation sublanguage describes the operations to be performed on the data without indication of the sequencing between operations, it answers to the question of HOW an operation is performed. The operations are subdivided into two groups.

- a computation group
- a data manipulation group

The later is the most important part of NIL since its main purpose is the transformation of data structures and patterns.

1.2.3 Data declaration sublanguage

The data declaration sublanguage is necessary to declare the variables and data structures on which operations are performed.

The possibility is given to build structures of atomic elements called beads. NIL provides a standard set of beads used in the "standard mode"; in the "extended mode" a user may define new beads and new structures of them.

1.2.4 Environment sublanguage

The environment sublanguage expresses the context in which a program expects to operate; expected characteristics of the peripherals, semantics of the exchanges with the outside world through a particular operating system.

Thus a complete "program descriptor" will contain four distinct sections:

environment section
data declaration section
control section
operation section

The identification section is omitted because it corresponds to the log in and socket grabbing part of the initialization procedure.

1.3 The Network Machine

One fundamental concept in NIL is that of an abstract Network Machine which has the following characteristics:

- an infinite memory: there is no problem of memory allocation or garbage collection in this machine. But as an item must be accessible, it must still have an address.
- variable word length: a word may be considered as the smallest intelligible and addressable item of data. The atomic element called bead is in fact the machine word. The structure and length of each type of beads are expressed in the data definition sublanguage.

As presented on figure 1.3.1, one HOST only communicates with a Network Machine which may operate in two modes.

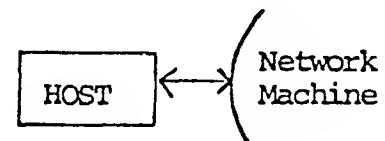


figure 1.3.1

- standard mode where the beads, ~~figure~~ their structures, and the allowed transformations on them are standard and need not to be redefined: standard beads and structures are known of every HOST
- extended mode where in addition to or instead of the standard data definitions and manipulation, a HOST may specify new beads structures and transformations. The extended mode allows the user to define his own machine as the Network Machine. This is then equivalent to the modes MY LOCAL, YOUR LOCAL proposed in RFC #42 by Ancona. If the definition of a name has not been altered the standard definition is assumed.

The data definition sublanguage is as well used for the purpose of documenting the set of standard beads.

The instruction set of the Network Machine stands at a high level permitting global transformations of data structures.

The environment of the Network Machine is determined by the subset of the environment of the server's HOST which is used by the program in execution; the system HOST-Network Machine can take two main configurations shown in figure 1.3.2.

- a) The Network Machine stands for the user of a program provided by the HOST (server HOST)
- b) The HOST machine is the user of a program provided by the Network Machine.

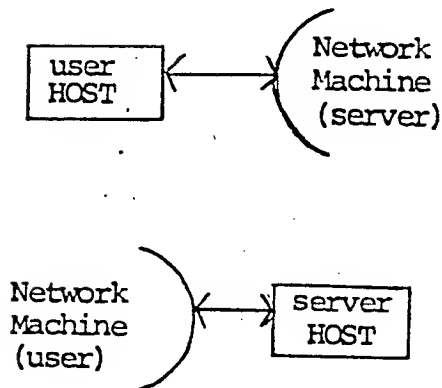


figure 1.3.2

The server machine assigns its hardware environment to the user machine. This choice is made so that programs can be remotely used without being modified; it is up to the user of a remote program to adapt himself and his own environment.

Thus, when the Network Machine is server, it defines the Data Definition and Environment sections.

1.4 Implementation

The data and environment definition sublanguages should be able to describe as well environment and data in HOSTs as data in Network Machine. At the limit it should enable two programs written in different languages to communicate,

as long as the data representation they use are expressible in the data description sublanguage.

In each HOST will be implemented a "generator" which will accept rules describing the HOST data structures and environment and will generate an adequate translator to translate them in Network Machine format, as shown in the figure 1.4.1.

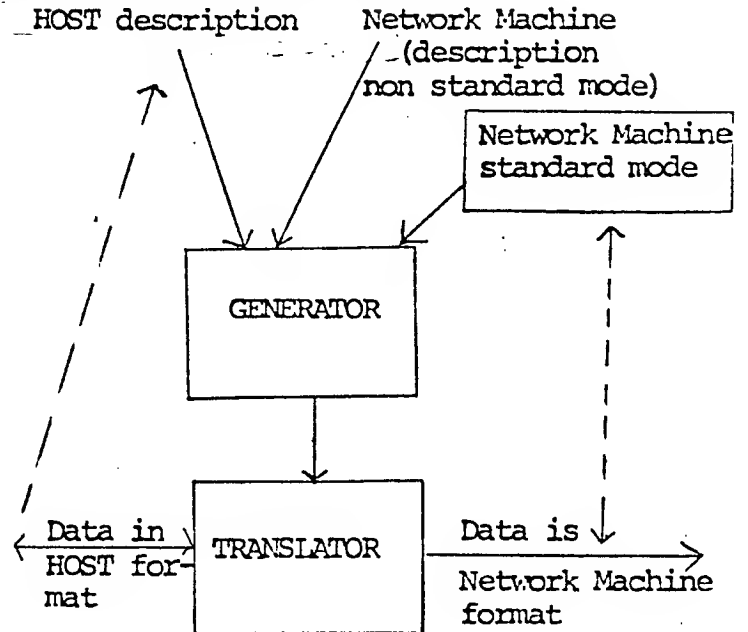


figure 1.4.1

Once the network machine standards will be settled it seems valuable to think about emulating the translator using a microprogrammed unit which would be either added to the Host or rather to the IMP" thus avoiding the load of a translation which may involve lengthy operations on the bit level - (Figure 1.4.2.)

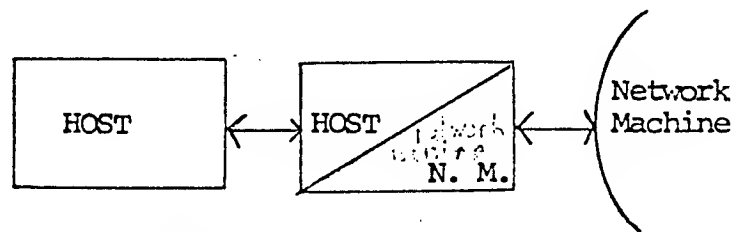


Figure 1.4.2

2. Data definition sublanguage

2.1 Fields

All communications with the Network Machine are done using strings of bits: these strings of bits, also referred to as messages are parsed by the receiving HOST to reconstruct inside its memory the data structures in its own memory and code.

Bits are grouped into significant fields: a field is a group of bits having definite contents. It may contain:

- a) an element of data (data field)
- b) some bit pattern specifying environment parameters
- c) a pointer
- d) the identification of some other fields.

The method to describe the formats of beads is derived from the method of description of a binary message suggested in RFC #31:

- a) each field is declared with its name and length in number of bits.
- b) commonly used fixed values of a field that correspond to a special meaning, may be given names.
- c) legal ways of concatenating fields are initiated by rules; when only certain fixed values of a field are allowed, they may be either specified by their value or by the corresponding name.

2.2 Data beads.

Data fields (type (a)) are concatenated to form data beads: a bead is an indivisible atomic unit of data used as building element of any data structure to be transmitted between HOSTs and Network Machine. A bead is the smallest unit of data that can be referenced.

The legal ways of forming a bead by concatenation of several fields are indicated in a construction rule. Beads have a fixed length and an unambiguous structure. In real machine beads are usually defined as an integer number of contiguous registers. This constraint does not apply here, though it may turn out to be more efficient to favor HOSTs with, for instance, 32 bit words, and 4 bytes per word, which are the most common word structure on the ARPA network.

Data beads may be considered as the operands of the language in which fields of type (b) and (c) would be operators.

2.3 Control fields

The way data beads are linked one to the other and the environment in which they operate are specified by additional control fields which cannot be referenced and are operators on or identifiers of the following string of beads, or linkage between individual beads.

The scope of a control field may as well be all the beads or substructures of a structure, if it is specified at the level of the head of the structure. To be more precise, two kinds of structures of beads must be defined: homogeneous and heterogeneous structures.

A structure is defined as homogeneous if both a unique type of bead and a fixed parameter environment for the whole structure is specified at the head of the structure.

A structure is defined as heterogeneous if at least one of the following conditions is true.

- different types of beads are used for building the structure
- the environment in which lie the beads of the structure is changing within the structure.

Five main control fields need to be defined.

- a) MODIFY
- b) FLAG
- c) POINTER
- d) IDENTIFICATION
- e) PARAMETER

2.3.1 MODIFY field

The MODIFY field is a one bit field preceding every bead of a heterogeneous structure: it is a flag set when followed by one or several control fields of type b, d, or e, which aim at modifying either the environment of data beads or their type. This field has the value:

- 1 if the attached data bead type and its environment do not change
- 0 if the attached element is a control field or a sequence of control fields of type b, d, or e specifying a change in type/or environment of following data beads.

2.3.2 FLAG field

When set, the MODIFY field is immediately followed by an 8 bit FLAG field indicating which of the IDENTIFICATION and several possible PARAMETER fields are present; when set to one each individual bit means the following:

bit number

0	IDENTIFICATION field present
1	first parameter field present
2	second parameter field present
6	sixth parameter field present
7	next field, is another FLAG field for some more parameters (in case more than 6 parameters may be attached to a bead environment).

2.3.3 POINTER field

The number and nature of pointers to be attached to each bead depends on the structure definition. A given list structure may need one forward pointer. A ring structure may use an additional pointer to the first element. The necessary linkage between beads are defined in the structure definition thereafter the necessary pointer fields automatically added to each data bead. A bead is referenced within a structure by an address relative to the head of the structure. Thus a 16 bit pointer field should be fully sufficient to contain this address.

2.3.4 IDENTIFICATION field

The IDENTIFICATION field is an 8 bit field which identifies a bead type among the list of defined bead types. Standard bead types are numbered from zero up and non-standard bead types are numbered from 255 down. The non-standard types numbering is special to each server program or to a set of server programs. The IDENTIFICATION fields follow a MODIFY field of value 1 whenever the bead type has not been defined all over the structure in the structure root. Identification fields are also used at the level of the head of the structure to specify the type of identical elements (beads or structures) used within this structure.

2.3.5 PARAMETER field

The PARAMETER field gives the list of the environment parameters in which the following string of data beads lies. A PARAMETER field is specific of a bead type; it directly follows the MODIFY field when there is no ambiguity or the type of the next data beads.

Example: the parameter field of the standard bead BEAMMVT will contain the following fields.

- a 2 bit field indicating the type of movement generated

00	do not display	move the beam
01	display final point	point
10	display vector	vector
11	unused	

- a 4 bit field indicating beam intensity, by a number from 0 to 15, 0 meaning a null intensity, and 15 the maximum possible intensity.

- a 1 bit field for blinking

0	off
1	on

- a 1 bit field for light pen sensitivity

0	off
1	on

2.4 Metalinguage definition

A COBOL - report like meta language is used in the examples because of its readability, as well in the beads as in the structure definitions.

Symbol	Meaning
+	concatanation
{ }	choice
[]	optional choice
{ } l <u>n</u> u	repetition

l lower bound on the number of identical items; if omitted l is assumed to be 0.

u upper bound on the number of identical items; if omitted u is assumed to be ∞.

a number alone means: exact number of repetition.

:	label for further use <u>within the same rule</u>
=	assignment
= >	conditional alternative
()	grouping
' '	indicates a special value given to the following field name
⊕	plus
⊖	minus

2.5 Proposed standard beads

2.5.1 Alphanumeric beads

Character: CHAR

A character is composed of one eight bit field (which has the same name). Many special patterns, corresponding to currently used special characters are defined; they are indicated in table 2.5.1, as well as some subsets of CHAR. The basic character code is declared as standard ASCII

standard EBCDIC

or by the name

CODE

followed by the 128 characters in this code corresponding to the 128 ASCII characters. If no code declaration is specified, the ASCII code is assumed by default.

Number representation

Normally the kernel of a program stays in the server's HOST and the user's HOST should have no arithmetic operations to perform on the data. In this case, the principles involved in the arithmetic unit conception of a HOST do not need to be described. But the format of fixed and floating point numbers has to be described.

- in the case when user and server HOST's have the same number representation, for instance the standard representation, the transmission of data in their number representation reduces the data flow between them.
- if the server HOST has a different number representation than the standard representation, depending on the data transmitted, there are two alternatives:

+ the numerical data is exchanged as decimal numbers in the standard code

+ the fixed and floating point format are defined to the Network Machine and the user HOST performs

either a direct transcoding from the server binary representation to decimal representation and vice versa.

or a transcoding from the server binary representation to its own binary representation and vice versa.

As most of the numbers exchanged are to be printed in decimal or are given as decimal input, it is felt that when there is incompatibility between binary representations of corresponding HOSTs, exchanges in decimal representation would be the easiest.

Thus are defined:

a) Number in decimal representation which is not a bead but a string of characters (see 2.3.1)

b) Fixed point numbers single precision FXPNUM1
 double precision FXPNUM2

Field definition	BYTE	8	SIGN	1
	SBYTE	7		

$$\text{FXP NUM1} \leftarrow \text{SIGN} + \text{SBYTE} + \{\text{BYTE}\}^3$$
$$\text{FXP NUM2} \leftarrow \text{FXPNUM 1} + \{\text{BYTE}\}^4$$

c) Floating point numbers single precision FLPNUM1
 double precision FLPNUM2

$$\text{FLP NUM1} \leftarrow \text{SIGN} + \text{SBYTE} + \{\text{BYTE}\}^3$$
$$\text{FLP NUM2} \leftarrow \text{FLPNUM1} + \{\text{BYTE}\}^4$$

This only expresses the syntax of the floating point number.
The semantics should say: in FLPNUM1

SIGN is the sign of the number of format $\{\text{BYTE}\}^3$ which is the mantissa

SBYTE is the exponent, and its value is based by a value of 40_{16} to insure positive exponents. In fact, FXPNUM1 and FLPNUM1 differ by their semantics.

These properties will be expressed by special field definition:

$$\begin{aligned} \text{EXP} &\leftarrow \text{SBYTE} \oplus '40\text{H}' \text{SBYTE} \\ \text{MANT} &\leftarrow \text{SIGN} \oplus \{\text{BYTE}\}^3 \end{aligned}$$

and a floating point number is defined as:

$$\text{FLP} = \text{MANT} \cdot 2^{\text{EXP}}$$

1. Special Characters

Transmission Control Characters

SOH
STX
ETX
EOT
ENQ
ACK
DLE
NAK
SYN
ETB
ESC

Printer Control Characters

horizontal tabulation	HT	+	'0x1' CHAR
vertical tabulation	VT	+	'0bX' CHAR
new line	NL	+	'0aX' CHAR
end of message	EOM	+	'08X' CHAR

Teletype Control Characters

Carriage return	CR	+	'0DX' CHAR
shift out	SO	+	'0eX' CHAR
shift in	SI	+	'0fX' CHAR
	BS	+	

Device Control Characters

DC1
DC2
DC3
DC4

Table 2.3.1

2. Subsets of Characters

Numeric characters		$\left\{ \begin{array}{c} 1 \\ 2 \\ \vdots \\ 9 \end{array} \right\}$	CHAR
NUM	+		
Printable characters		$\left\{ \begin{array}{c} \text{NUM} \\ \text{ALPH} \\ \text{=} \\ \text{=} \end{array} \right\}$	CHAR
PRCHAR	+		
Intermediate characters		$\left\{ \begin{array}{c} \text{'characters'} \\ \text{in column} \\ 2 \end{array} \right\}$	CHAR
ITCHAR	+		
Final Characters			
FIN CHAR	+	CHAR	$\ominus$ ITCHAR
Transmission Control Characters*		$\left\{ \begin{array}{c} \text{'NUL'} \\ \vdots \end{array} \right\}$	CHAR
TRACHAR	+		
Delta Control Characters		DEL	Teletype control character
DCCHAR	+	$\left\{ \begin{array}{c} \text{'DC1'} \\ \text{'DC2'} \\ \text{'DC3'} \\ \text{'DC4'} \end{array} \right\}$	CHAR TYCCHAR $\left\{ \begin{array}{c} \text{'CR'} \\ \text{'SO'} \\ \text{'SI'} \\ \text{'BS'} \end{array} \right\}$
Alphabetic characters		$\left\{ \begin{array}{c} \text{'A'} \\ \vdots \\ 2 \end{array} \right\}$	CHAR
ALPH	+		
Printer Control Characters		$\left\{ \begin{array}{c} \text{'HT'} \\ \text{'VT'} \\ \text{'NL'} \\ \text{'EQM'} \end{array} \right\}$	CHAR
POCHAR	+		

Table 2.3.1: Special ASCII characters and groups of ASCII characters.

*see USASCII standards

2.5.2 Graphic Beads

As proposed in RFC #5 by J. Rulifson, the screen of any graphical display is taken to be a square; the coordinates of points are normalized from $-1/2$ to $+1/2$ on both axes. The position of the first point of a structure is determined by the deflection from the origin which is the rest point of the beam; following points are determined by their deflections (AX,AY) from the last beam position.

Thus, only two data fields need to be defined:

DEFLECTION	which is a 12 bit field: the deflection is defined by a number between - 1 and +1 with the precision usual to the server.
ANGLE	which is a 15 bit field defining an angle from 0 to 2π in radians between the horizontal axis and an axis passing through the origin. The first bit of it indicates if the angle must be taken clockwise or counterclockwise.

The data beads are:

MOVE

Depending on the parameters which are set when this bead appears, MOVE may specify:

- an invisible movement of the beam; in this case the beam intensity is null
- a new point: in this case the beam intensity is on only when the beam has reached the new point.
- a vector: in this case the beam intensity is set to a certain non zero value

$\text{MOVE} + \{\text{DEFLECTION}\}^2$

Arc of circle: ARC

An arc of circle is defined by its center, followed by its starting point and the angle of its ending axis.

$\text{ARC} + \{\text{DEFLECTION}\}^4 + \text{ANGLE}$

2.6 Proposed parameter fields.

2.6.1 Character strings.

In character strings some of the control characters are really parameter fields: they act as an operator on the following string of characters. i.e.:

lower shift
upper shift
new line
escape
.....

But as the code and use of these characters are determined in the standard codes, they are not included in parameter definition. It may be taken advantage that these characters are in the two left columns of the ASCII or EBCDIC standard code: they correspond to codes with the first three bits null in EBCDIC and the first two bits null in ASCII.

2.6.2 Graphics parameters

The following parameter fields are defined:

scale	SCALE	4
beam intensity	INT	4
light pen sensitivity	SENS	+ SWITCH
blinking	BLINK	+ SWITCH
beam	BEAM	+ SWITCH

SWITCH is a 1 bit field which may take the values:

ON ← '1' SWITCH
OFF ← '0' SWITCH

A switch parameter stays ON, as long as it is not reset to OFF.

The beam intensity is expressed by a number from 0 to 1. 0 is black and 1 as light as the display can go. Numbers in between specify the relative log of the intensity difference. BEAM permits to switch the BEAM on or off without changing the Current INT parameter.

2.7 Structures

2.7.1 Structure definition.

The structure definition consists mainly in the specification of the topological relations between data nodes:

- sequential relations; no pointer field necessary
- links through a number of pointers.

2.7.2 Standard structure type.

Two basic standard structure types are chosen

VECTOR to represent sequence of data beads (strings, arrays, tables...)

PLEX to represent any kind of directed graph, tree, ring...)

VECTOR (C;N,...N_C) ← VECTORHDR + VECTORBODY

VECTORBODY ← (=C+1:{defined bead}^{N_i} + [VECTORBODY]

VECTORHDR ← 'VECTOR' IDENTIFICATION + C + N₁ + N₂ + ... + N_C
C is the number of parts (columns) in the vector, each part having N_C elements.

It is also probably interesting to define a compressed vector COMPVECTOR in which sequence of the identical elements are transmitted as 1 element + a special bead + the number of identical elements in sequence.

PLEX (M)

The first bit of a pointer field indicates if the pointer points to a terminal element or not. If it is the case, forward pointer fields are not added to the data element.

M is the number of data elements in the structure.

2.8 Objects

2.8.1 object definition.

An object is defined by a semantic rule including, on the right hand side {

a name to identify the object
a set of parameters of the object definition.

on the left { operands: name of the beads used as data elements

hand side { operators: parameter fields
structure of the data beads.

i.e. The definition of a new object called SQUARE is: SQUARE ←
(A,L;A) ROT(A;ANGLE) (VECTOR(1,4) (BEAM'OFF'+
MOVE (A) + BEAM 'ON' + MOVE (0,L) + MOVE (L,0) +
MOVE (0,eL) + MOVE (eL,0))

Where ROT refers to a transformation defined in the data manipulation language, and VECTOR is defined as a standard structure.

The identifier of the new structure is SQUARE
The structure type used is VECTOR with dimension 1 and 4 elements
The elements of the VECTOR are standard beads MOVE
The parameters are A, L, and A 0
Parameter fields BEAM 'OFF' and BEAM 'ON' are used.

2.8.2 Alphanumeric standard objects.

Compressed character string (COMSTRING)

$$\text{COMSTRING} \leftarrow \text{VECTOR} (1) \left(\left\{ \left[\text{PRCHAR} \right]^n + \begin{bmatrix} \text{HT+NUM} \\ \text{VT+NUM} \\ \text{ESC+CHAR} \\ \text{NL} \\ \text{EOP} \end{bmatrix} \right\}^n \text{ EOF} \right)$$

A compressed string of characters is any number of times a string of any number of printable characters followed by one of the following characters

- horizontal tabulation followed by the number of corresponding blanks to be added
- vertical tabulation followed by the number of lines to be skipped.
- escape followed by any character
- new line
- end of page

The compressed string is ended by an EOF character.

- Code table (CODE)
CODE ← VECTOR (1;128) {CHAR} ¹²⁸

CODE is the name of the translation table assumed for a given program. When defined by the user, he must give from column 1 to column 8 the 8 bit pattern equivalent to the corresponding ASCII code.

- Binary card image

B CARD ← VECTOR (1;120) {CHAR}¹²⁰

Packed decimal number HNUM 4 bits field

$$\begin{array}{c}
 \left\{ \begin{array}{cc} A & X \\ C & X \\ E & X \\ F & X \\ B & X \\ D & X \end{array} \right\} \text{ HNUM} \quad \text{PNUM} \leftarrow \left\{ \begin{array}{c} O \text{ H} \\ \vdots \\ 9 \text{ H} \end{array} \right\} \text{ HNUM} \\
 \text{PDNUM} \leftarrow \left\{ \text{PNUM} \right\}^{1 \leq n \leq 31} + \text{DSIGN}
 \end{array}$$

- Decimal number (unpacked or zoned

$$\text{DNUM} \leftarrow \left\{ \text{NUM} \right\}^{1 \leq n \leq 31} + \text{D SIGN} + \text{PNUM}$$

Postel

Title: DEL

Author: Jeff Rulifson

Installation: Stanford Research Institute

Date: 2 June 1969

Network Working Group Request for Comment: 5

:DEL, 02/06/69 1010:58 JFR ; .DSN=1; .LSP=0; [=] AND NOT SP ; [?];
dual transmission?

Abstract.

The Decode-Encode Language (DEL) is a machine independent language tailored to two specific computer network tasks:

accepting input codes from interactive consoles, giving immediate feedback, and packing the resulting information into message packets for network transmission.

and accepting message packets from another computer, unpacking them, building trees of display information, and sending other information to the user at his interactive station.

This is a working document for the evolution of the DEL language. Comments should be made through Jeff Rulifson at SRI.

Foreword.

The initial ARPA network working group met at SRI on October 25-26, 1968.

It was generally agreed beforehand that the running of interactive programs across the network was the first problem that would be faced.

This group, already in agreement about the underlying notions of a DEL-like approach, set down some terminology, expectations for DEL programs, and lists of proposed semantic capability.

At the meeting were Andrews, Baray, Carr, Crocker, Rulifson, and Stoughton.

A second round of meetings was then held in a piecemeal way.

Crocker met with Rulifson at SRI on November 16, 1968. This resulted in the incorporation of formal co-routines.

and Stoughton met with Rulifson at SRI on December 12, 1968. It was decided to meet again, as a group, probably at UTAH, in late January, 1969.

The first public release of this paper was at the BBN NET meeting in Cambridge on February 13, 1969.

NET Standard Translations.

NSI The NSI library is the set of programs necessary to mesh efficiently with the code compiled at the user sites from the DEL programs it receives. The NSI-DEL approach to NET interactive system communication is intended to operate over a broad spectrum.

The lowest level of NST-DEL use would be to have the server-host send information in the same format that the user would receive at the user-host.

In this mode, the NST default character set is used. The DEL program does not receive universal hardware representation of input in the normal fashion for the user-host.

And the DEL program becomes merely a display builder and sender.

A more intermediate use of NST-DEL is to have echo tables for a TTY at the user-host.

In this mode, the DEL program would run a full duplex TTY for the user.

It would echo characters, translate them to the character set of the server-host, pack the translated characters in messages, and on appropriate break characters, send the messages.

When messages come from the server-host, the DEL program would translate them to the user-host character set and print them on his TTY.

A more ambitious task for DEL is the operation of large, display-oriented systems from remote consoles over the NET.

Large interactive systems usually offer a lot of feedback to the user. The unusual nature of the feedback make it impossible to model with echo tables, and thus a user program must be activated in a TSS each time a button state is changed.

This puts an unnecessarily large load on a TSS, and if the system is being run through the NET it could easily load two systems.

To avoid this double overloading of TSS, a DEL program will run on the user-host. It will handle all the immediate feedback, much like a complicated echo table. At appropriate button pushes, message will be sent to the server-host and display updates received in return.

One of the more difficult, and often neglected, problems is the effective simulation of one non-standard console on another non-standard console.

We attempt to offer a means of solving this problem through the co-routine structure of DEL programs. For the complicated interactive systems, part of the DEL programs will be constructed by the server-host programmers. Interfaces between this program and the input stream may easily be inserted by programmers at the user-host site.

Universal Hardware Representation

To minimize the number of translators needed to map any facility's user codes to any other facility, there is a universal hardware representation.

This is simply a way of talking, in general terms, about all the hardware devices at all the interactive display stations in the initial network.

For example, a display is thought of as being a square, the mid-point has coordinates (0,0), the range is -1 to 1 on both axes. A point may now be specified to any accuracy, regardless of the particular number or density of raster points on a display.

The representation is discussed in the semantic explanations accompanying the formal description of DEL.

Introduction to the Network Standard Translator (NST).

Suppose that a user at a remote site, say Utah, is entered in the AHI system and wants to run NLS.

The first step is to enter NLS in the normal way. At that time the Utah system will request a symbolic program from NLS.

REP This program is written in DEL. It is called the NLS Remote Encode Program (REP).

The program accepts input in the Universal Hardware Representation and translates it to a form usable by NLS.

It may pack characters in a buffer, also do some local feedback.

When the program is first received at Utah it is compiled and loaded to be run in conjunction with a standard library.

All input from the Utah console first goes to the NLS NEP. It is processed, parsed, blocked, translated, etc. When NEP receives a character appropriate to its state it may finally initiate transfers to the 940. The bits transferred are in a form acceptable to the 940, and maybe in a standard form so that the NLS need not differentiate between Utah and other NET users.

Advantages of NST:

After each node has implemented the library part of the NST, it need only write one program for each subsystem, namely the symbolic file it sends to each user that maps the NET hardware representation into its own special bit formats.

This is the minimum programming that can be expected if each

console is used to its fullest extent.

Since the NST which runs the encode translation is coded at the user site, it can take advantage of hardware at its consoles to the fullest extent. It can also add or remove hardware features without requiring new or different translation tables from the host.

Local users are also kept up to date on any changes in the system offered at the host site. As new features are added, the host programmers change the symbolic encode program. When this new program is compiled and used at the user site, the new features are automatically included.

The advantages of having the encode translation programs transferred symbolically should be obvious.

Each site can translate any way it sees fit. Thus machine code for each site can be produced to fit that site; faster run times and greater code density will be the result.

Moreover, extra symbolic programs, coded at the user site, may be easily interfaced between the user's monitor system and the DEL program from the host machine. This should ease the problem of console extension (e.g. accommodating unusual keys and buttons) without loss of the flexibility needed for man-machine interaction.

It is expected that when there is matching hardware, the symbolic programs will take this into account and avoid any unnecessary computing. This is immediately possible through the code translation constructs of DEL. It may someday be possible through program composition (when Crocker tells us how??).

AHI NLS - User Console Communication - An Example.

Block Diagram

The right side of the picture represents functions done at the user's main computer; the left side represents those done at the host computer.

Each label in the picture corresponds to a statement with the same name.

There are four trails associated with this picture. The first links (in a forward direction) the labels which are concerned only with network information. The second links the total information flow (again in a forward direction). The last two are equivalent to the first two but in a backward direction. They may be set with pointers t1 through t4 respectively.

[">tif"] OR [">nif"]; ["<tif"] OR ["<nif"];

User-to-Host Transmission

keyboard is the set of input devices at the user's console. Input bits from stations, after drifting through levels of monitor and interrupt handlers, eventually come to the encode translator. [>nif(encode)]

encode maps the semi-raw input bits into an input stream in a form suited to the serving-host subsystem which will process the input. [>nif(hrt) <nif(keyboard)]

The Encode program was supplied by the server-host subsystem when the subsystem was first requested. It is sent to the user machine in symbolic form and is compiled at the user machine into code particularly suited to that machine.

It may pack to break characters, map multiple characters to single characters and vice versa, do character translation, and give immediate feedback to the user.

Immediate feedback from the encode translator first goes to local display management, where it is mapped from the NET standard to the local display hardware.

A wide range of echo output may come from the encode translator. Simple character echoes would be a minimum, while command and machine-state feedback will be common.

It is reasonable to expect control and feedback functions not even done at the server-host user stations to be done in local display control. For example, people with high-speed displays may want to selectively clear curves on a Culler display, a function which is impossible on a storage tube.

Output from the encode translator for the server-host goes to the invisible IMP, is broken into appropriate sizes and labeled by the encode translator, and then goes to the NET-to-host translator.

Output from the user may be more than on-line input. It may be larger items, such as computer-generated data, or files generated and used exclusively at the server-host site but stored at the user-host site.

Information of this kind may avoid translation, if it is already in server-host format, or it may undergo yet another kind of translation if it is a block of data.

It finally gets to the host, and must then go through the host reception program. This maps and reorders the standard transmission-style packets of bits sent by the encode programs into messages acceptable to the host. This program may well be

part of the monitor of the host machine. [>tif(net
mode)<nif(encode)]

Host-to-User Transmission

decode Output from the server-host initially goes through decode, a translation map similar to, and perhaps more complicated than, the encode map. [>nif(urt)>tif(imp ctrl)<tif(net mode)]

This map at least formats display output into a simplified logical-entity output stream, of which meaningful pieces may be dealt with in various ways at the user site.

The Decode program was sent to the host machine at the same time that the Encode program was sent to the user machine. The program is initially in symbolic form and is compiled for efficient running at the host machine.

Lines of characters should be logically identified so that different line widths can be handled at the user site.

Some form of logical line identification must also be made. For example, if a straight line is to be drawn across the display this fact should be transmitted, rather than a series of 500 short vectors.

As things firm up, more and more complicated structural display information (in the manner of LEAP) should be sent and accommodated at user sites so that the responsibility for real-time display manipulation may shift closer to the user.

imp ctrl The server-host may also want to send control information to IMPs. Formatting of this information is done by the host decoder. [>tif(urt) <tif(decode)]

The other control information supplied by the host decoder is message break up and identification so that proper assembly and sorting can be done at the user site.

From the host decoder, information goes to the invisible IMP, and directly to the NET-to-user translator. The only operation done on the messages is that they may be shuffled.

urt The user reception translator accepts messages from the user-site IMP and fixes them up for user-site display. [>nif(d ctrl)>tif(prgm ctrl)<tif(imp ctrl)<nif(decode)]

The minimal action is a reordering of the message pieces.

dctrl For display output, however, more needs be done. The NET logical display information must be put in the format of the user site. Display control does this job. Since it coordinates between (encode) and (decode) it is able to offer

features of display management local to the user site. [>nif(display) <nif(urt)]

prgmctrl Another action may be the selective translation and routing of information to particular user-site subsystems. [>tif(d ctrl) <tif(urt)]

For example, blocks of floating-point information may be converted to user-style words and sent, in block form, to a subsystem for processing or storage.

The styles and translation of this information may well be a compact binary format suitable for quick translation, rather than a print-image-oriented format.

(display) is the output to the user. [<nif(d ctrl)]

User-to-Host Indirect Transmission

(net mode) This is the mode where a remote user can link to a node indirectly through another node. [>tif(decode) <tif(hrt)]

DEL Syntax.

Notes for NLS Users.

All statements in this branch which are not part of the compiler must end with a period.

To compile the DEL compiler:

Set this pattern for the content analyzer (*PI SE(PI) < -"_:). The pointer "del" is on the first character of pattern.

Jump to the first statement of the compiler. The pointer "c" is on this statement..

And output the compiler to file('/A-DEL'). The pointer "f" is on the name of the file for the compiler output.

Programs.

Syntax.

_meta file (k=100, m=300, n=20, s=900)

file = mesdecl \$declaration \$procedure "FINISH";

procedure =

 procname (

 i

```

type "FUNCTION" /
      "PROCEDURE" ) .id?$(type .id / .empty)) /
"CO-ROUTINE") "; /

```

```

$declaration labeledst $(labeledst ";") "endp.";

```

```

labeledst = (".id "; / .empty) statement;

```

```

type = "INTEGER" / "REAL";

```

```

procname = .id;

```

Functions are differentiated from procedures to aid compilers in better code production and run time checks.

Functions return values.

Procedures do not return values.

Co-routines do not have names or arguments. Their initial invocation points are given the pipe declaration.

It is not clear just how global declarations are to be??

Declarations.

Syntax.

```

declaration = numbertype / structuredtype / label / lc|2uhr /
uhr2rmt / pipetype;

```

```

numbertype = : ("REAL" / "INTEGER") ("CONSTANT" conlist /
varlist);

```

```

conlist =

```

```

    .id "+ constant

```

```

    $(" .id "+ constant);

```

```

varlist =

```

```

    .id (" + constant / .empty)

```

```

    $(" .id (" + constant / .empty));

```

```

idlist = .id $(" .id);

```

```

structuredtype = ("tree" / "pointer" / "buffer" ) idlist;

```

```

label = "LABEL" idlist;

```



```

/* \ orlop /
/* \ orlop /
bilop = compilation (
factor = " - factor / bilop:
- empty:
/* term /
/* term /
/* term /
term = factor:
- empty:
- sum /
+ sum /
sum = term:
exp = "if" conjunct "THEN" exp "ELSE" exp:

```

Syntax

Arithmetic

Variables which are declared to be constant, may be put in read-only memory at run time.

The label declaration is to declare cells which may contain the machine addresses of labels in the program as their values. This is not the B5500 label declaration.

In the pipe declaration the first ID of each pair is the name of the pipe, the second is the initial starting point for the pipe.

```

label = id:
pipename = id:
integer = id:
procname = id:
pairedids = id id:
pipe type = "PIPE" pairedids s(" pairedids):

```

constant /

message /

builtin =

constant = integer / real / string;

-empty::

" { block } " /

" - exp /

variable = - id {

" { exp } " /

block /

variable /

builtin /

constant /

primary =

Syntax-

Primary-

It is assumed that all arithmetic and bit operations take place in the mode and style of the machine running the code. Anyone who takes advantage of word lengths, two's complement arithmetic, etc. will eventually have problems.

Complement is the 1's complement.

grouping-

Since there is no standard convention with bitwise operators, they all have the same precedence, and parentheses must be used for

write $x \sim y$.

Notice that the unary minus is allowable, and parsed so you can

* means mod, and / means exclusive or.

complement = " ~ " primary / primary;

-empty::

" { block } "

("MIN" / "MAX") exp \$("exp" "\ ;

parenthesised expressions may be a series of expressions. The value of a series is the value of the last one executed at run time.

Subroutines may have one call by name argument. ?

Expressions may be mixed. Strings are a big problem?? Rutliffson also wants to get rid of real numbers!!

Conjunctive Expression.

Syntax.

conjunct = disjunct ("AND" conjunct / .empty);

disjunct = negation ("OR" negation / .empty);

negation = "NOT" relation / relation;

relation =

"(" conjunct ")" /

sum {

"<" sum /

">" sum /

"<" sum /

">" sum /

"=" sum /

"#" sum /

.empty);

The conjunct construct is rigged in such a way that a conjunct which is not a sum need not have a value, and may be evaluated using jumps in the code. Reference to the conjunct is made only in places where a logical decision is called for (e.g. if and while statements).

We hope that most compilers will be smart enough to skip unnecessary evaluations at run time. I.e. a conjunct in which the left part is false or a disjunct with the left part true need not have the corresponding right part evaluated.

Arithmetic Expression.

```

    replacename = "replace" pname "with" ptrexp;

    plantree = "plant" tree "in" treearea;

    tail;

    head /

    backward /

    forward /

    down /

    up /

    direction =

    ((first / last) "daughter") "of" ptrexp;

    ((left / right) "brother") /

    insertptr = "insert" ptrexp "as"

    ptrexp = direction ptrexp / ptrname;

    setptr = "set" "pointer" ptrname "to" ptrexp;

    freest = setptr / insertptr / deleteptr;

```

Syntax-

Semi-tree Manipulation and Testing-

ordered-
is a side effect of the way the left part of the syntax rules are
else part is optional while in expressions is mandatory. This
An expressions may be a statement. In conditional statements the

```

    block = "begin" exp "(" exp) "end";

    - empty;

    "ELSE" conditional /

    conditional = "IF" conjunct "THEN" unconditional {

    unconditional = loop / case / control / lost / freest /
    block / null / exp;

```

Syntax-

```

forst = "FOR" integerV "+ exp ("BY" exp / -empty) "TO" exp
untilst = "UNTIL" conjunct "DO" statement;
whilest = "WHILE" conjunct "DO" statement;
loopst = whilest / untilst / forst;

```

Syntax.

Loop Statements.

FETCH is a built-in function whose value is computed by invoking the named subroutine.

```

contin = "FETCH" pipename;
continout = "STUFF" exp "IN" pipename;
returnst = "RETURN" (exp / -empty);
callst = "CALL" procname (exp / -empty);
subst = callst / returnst / continout;

```

Subroutines.

```

assignlabel = "ASSIGN" .id "TO" label;
gost = "GO" "TO" (label / .id);

```

Go To Statements.

```

controlst = gost / subst / loopst / castst;

```

Flow and Control.

Extra parentheses in tree building results in linear subcategorization, just as in LISP.

```

treedec = "pointer" .id / "tree" .id;
treename = .id;
terminal = treename / buffername / pointername;
nodename = terminal / " ( tree " ) ;
tree = nodename $nodename ;
tree = " ( tree " ) ;
deleteptr = "delete" ptrname;

```

The value of while and until statements is defined to be false and true (or 0 and non-zero) respectively.

For statements evaluate their initial exp. by parts and to pass once at initialization time. The running index of for statements is not available for change within the loop, it may only be read. If some compilers can take advantage of this (say put it in a register) all the better. The increment and the to bound will both be rounded to integers during the initialization.

Case statements:

Syntax-

```
casest = ithcasest / condcasest;
```

```
ithcasest = "ITHCASE" exp "OF" "BEGIN" statement st';  
statement) "END";
```

```
condcasest = "CASE" exp "OF" "BEGIN" condcs st'; condcs)  
"OTHERWISE" statement "END";
```

```
condcs = "conjunct" : statement;
```

The value of a case statement is the value of the last case executed.

Extra statements.

```
null = "NULL";
```

I/O statements.

```
lost = messages / dspst ;
```

Messages-

Syntax-

```
messages = buildmes / demand;
```

```
buildmes = startmes / appendmes / sendmes;
```

```
startmes = "start" "message";
```

```
appendmes = "append" "message" "byte" exp ;
```

```
sendmes = "send" "message";
```

```
demandmes = "demand" "message";
```

The screen is taken to be a square. The coordinates are normalized from -1 to +1 on both axes.

Logical Screen -

estab = "establish" buffername;

onoff = "on" / "off";

"italics" onoff;

"blink" onoff /

"character" "width" "to" exp /

"intensity" "to" exp /

dsyparm =

"curve" ;

"position" (onoff / empty) "beam" "to" exp "exp" /

"vector" ("from" exp "exp" / empty) "to" exp "exp" /

"string" string /

"character" exp /

"parameters" dsyparm \$("dsyparm) /

bufstuff =

b <

bufappend = "append" bufstuff \$("b bufstuff);

startbuffer = "start" ~~buffer~~ clear(b)

dsypst = startbuffer / bufappend / ;

Syntax -

Display Buffers -

mesdel = "message" "bytes" "are" "num" "bits" "long" ; ;

"message" "empty" ; ;

"message" "length" /

"get" "message" "byte"

mesinfo =

Program to run display and keyboard as tty.

Sample Programs

end

Audio Output Devices

Mice

Light Pens

Buttons

Keyboard

Joy Stick

Hand

Logical Input Devices

tree building statement

As the buffer is built, the logical entities are added to it. When it is established as a buffername, the buffer is closed, and further appends are prohibited. It is only a buffername has been established that it may be used in a

When the buffer is initialized, it is empty. If no parameters are initially appended, those in effect at the end of the display of the last node in the semi-tree will be in effect for the display of this node.

The terminal nodes of semi-trees are either semi-tree names or display buffers. A display buffers is a series of logical entities, called buffers.

Buffer Building

Blink bit

Character frame size

The intensity, called INTENSITY, is a real number in the range from 0 to 1. 0 is black, 1 is as light as your display can go, and numbers in between specify the relative log of the intensity difference.

Associated with the screen is a position register, called PREC. The register is a triple (x, y, r), where x and y specify a point on the screen and r is a rotation in radians, counter clockwise, from the x-axis.

to run NLS-

input part

display part

DEMAND MESSAGE:

while LENGTH = 0 DO

!THCASE CEIBYTE OF Begin

!THCASE ICEIBYTE OF Zfile area updated BEGIN

Zlateral areaZ

Zmess age areaZ

Zname areaZ

ZoungZ

Zsequence specsz

Zfilter specsz

Zformat specsz

Zcommand feedback lineZ

Zfile areaZ

Zdate timeZ

Zcno registerZ

BEGIN MODEL controlZ

Distribution list

Steve Carr

Department of Computer Science

University of Utah

Salt Lake City, Utah 84112

Phone 801-322-3211 X8224

Steve Crocker

315 Boylston Hall

12-1-71
316 501-11-517
C. G. X

University of California

Los Angeles, California 90024

Phone 213-825-4864

314 7857

Jeff Kulifson

Stanford Research Institute

333 Ravenswood

Menlo Park, California 94305

Phone 415-326-6200 X4116

Ron Sloughton

Computer Research Laboratory

University of California

Santa Barbara, California 93106

Phone 805-961-3221

Mehmet Baray

Corey Hall

University of California

Berkeley, California 94720

Phone 415-843-2521